Exercice 1

1. Écrire la fonction récursive **calculerPuissance** à 2 paramètres entiers **x** et **y** qui retourne x^y

   Rappel : $x^y = x * x^{y-1}$

2. Faire la trace des appels pour **calculerPuissance**(4,3)

3. Écrire un programme qui demande à l'utilisateur de saisir 2 entiers **a** et **b** avec **a** non nul et **b** positif ou nul (il faudra vérifier les différentes saisies sinon on les recommence), puis calcule et affiche la puissance a^b. On utilisera la fonction précédente.

Exercice 2

1. Écrire la fonction récursive **calculerPgcd** à 2 paramètres **a** et **b** de type entier qui retourne le Plus Grand Commun Diviseur de **a** et de **b** . On supposera que **a** et **b** sont positifs.

   $$Rappel : Pgcd(a,b) = \begin{cases} a & \text{si b = 0} \\ Pgcd(b, a\%b) & \text{sinon} \end{cases}$$

2. Faire la trace des appels pour **calculerPgcd**(4,3)

3. Écrire la fonction **calculerPpcm** à 2 paramètres **a** et **b** de type entier qui retourne le Plus Petit Commun Multiple des 2 entiers **a** et **b**. On supposera que **a** et **b** sont positifs.

   Rappel : **calculerPpcm**(a, b) * **calculerPgcd**(a, b) = a * b

4. Écrire un programme qui demande à l'utilisateur de saisir 2 entiers **x** et **y** positifs puis affiche le **pgcd** et le **ppcm** des 2 entiers **x** et **y**. On utilisera les fonctions précédentes.

Exercice 3

1. Écrire la fonction récursive **calculerSomme** à 2 paramètres **a** et **b** de type entier qui retourne la somme a+b en utilisant la méthode suivante (attention aux cas des valeurs négatives) :

   $$Somme(a,b) = \begin{cases} b & \text{si a = 0} \\ Somme(a-1, b+1) & \text{sinon} \end{cases}$$

2. Faire la trace des appels de calculerSomme(4,5)

3. Écrire un programme qui demande à l'utilisateur de saisir 2 entiers **x** et **y** puis calcule et affiche la somme x+y. On utilisera la fonction précédente.

Exercice 4

1. Écrire la fonction récursive **calculerFibonacci** à un paramètre **n** de type entier qui retourne le nième terme de la suite de Fibonacci. On supposera que **n** est positif non nul.

   $$Rappel : Fibo(n) = \begin{cases} 1 & \text{si n = 1} \\ 1 & \text{si n = 2} \\ Fibo(n-1) + Fibo(n-2) & \text{n > 2} \end{cases}$$

2. Faire la trace des appels de **calculerFibonacci**(6)

3. Écrire un programme qui demande à l'utilisateur de saisir un entier positif non nul **n** puis qui calcule et affiche les nième termes de la suite de Fibonacci. On utilisera la fonction précédente.

```
Exemple : Entrer un entier positif non nul : 10
La suite de Fibonacci(10) :  1 1 2 3 5 8 13 21 34 55
```

Exercice 5

1. Écrire la fonction récursive **calculerProduit** à 2 paramètres **a** et **b** de type entier qui retourne le produit a*b en utilisant que des additions et soustractions. Attention aux cas des valeurs négatives.
2. Faire la trace des appels de **calculerProduit**(5,4)
3. Écrire un programme qui demande à l'utilisateur de saisir 2 entiers **x** et **y** qui calcule et affiche le produit x*y en utilisant la fonction précédente.
4. Faire le calcul avec le produit : (a) 1000 x 1, puis (b) 1 x 1000 : Que se passe-t-il ?
5. Améliorer le programme pour éviter ce problème.

Exercice 6

1. Écrire la fonction récursive **calculerMultiple** à 2 paramètres **a** et **b** de type entier qui retourne Vrai si **a** est multiple de **b** sinon Faux, en utilisant que des soustractions. On supposera que **a** et **b** sont positifs.
2. Faire la trace des appels de **calculerMultiple**(100, 20) et de **calculerMultiple**(100,21)
3. Écrire un programme qui demande à l'utilisateur de saisir 2 entiers **x** et **y** positifs puis affiche "x est un multiple de y" ou "x n'est pas un multiple de y" selon le cas. On utilisera la fonction précédente.

Exercice 7

1. Écrire la fonction récursive **calculerSommeN** à un paramètre **n** de type entier qui retourne la somme des **n** premiers entiers naturels. On supposera que **n** est positif.
2. Faire la trace des appels de **calculerSommeN**(5)
3. Écrire un programme qui saisit un entier positif **x** puis calcule et affiche la somme des **x** premiers entiers.

Exercice 8

1. Écrire fonction récursive **chercherElement** à 3 paramètres **t** de type tableau de 10 entiers, **x** de type entier et **r** de type entier qui retourne Vrai si **x** appartient à **t** sinon Faux. **r** représente la taille du domaine de recherche restant.
2. Faire la trace des appels de **chercherElement** avec :
   (a) t = (1, 2, 4, 5, 10, 12, 6, 7, 8, 11) et x=4
   (b) t = (1, 2, 3, 6, 17, 10, 12, 11, 30, 20) et x=14
3. Écrire une fonction récursive **compterOccurrences** à 3 paramètres **t**, de type tableau de 10 entiers, **x** de type entier et **r** de type entier, qui retourne le nombre d'occurrence de **x** dans le tableau **t**.
4. Faire la trace des appels de **compterOccurrences** avec :
   (a) t = (1, 2, 14, 5, 10, 6, 7, 8, 11, 12) et x=4
   (b) t = (1, 2, 3, 4, 3, 5, 6, 3, 3, 8) et x=3
5. Écrire un programme qui saisit un tableau **v** de 10 entiers, l'affiche, saisit 2 entiers **a** et **b**, affiche si **a** appartient à **v** et affiche le nombre d'occurrences de **b** dans **v**.

Exercice 9

1. Écrire fonction récursive **estPalindrome** qui retourne Vrai si une chaîne de caractères passée en paramètre est un palindrome sinon faux.
2. Écrire un programme qui demande la saisie d'une chaîne de caractères et qui indique si c'est un palindrome : par exemple : "'laval' est un palindrome", ou "'maison' n'est pas un palindrome".

   Rappel : si ch est de type string, ch[0] donne accès au 1er caractère de la chaine, ch.length() retourne la longueur de la chaine, ch.substr(po, lo) crée une sous-chaine de ch à partir du caractère de numéro po sur une longueur lo.